Building a REST API Server with PHP

Project: Building a User Management API Server

- GET /students - Get all students
- GET /students/{id} - Get student by ID
- POST /students - Create new student
- PUT /students/{id} - Update student
- DELETE /students/{id} - Delete student

This is an example, so we can add any additional APIs, for example:

- GET/POST /count - count the total number of students.
- GET/POST /search/{name} - search for student with the name {name}

In this case, we can use GET/POST or both in PHP:

- GET is easier to use
- POST is used with a form

Project Structure

```
api/
├── index.php                    # Main entry point and routing
├── handlers.php                 # All handler functions
├── models/Students.php          # Student model class
├── data/students.json           # JSON data storage
├── test.html                   # Web testing interface
└── start_server.sh/.bat        # Server startup scripts
```

In this example, for simplicity, we use JSON (data/students.json) for storing student information, however in a real project, students should use MySQL database.

index.php

- Header functions
- Input parser
- routers

Header functions

- We use these header() functions to configure how the server responds to API requests — including content type and cross-origin access.

```
header('Content-Type: application/json');
```

- Tells the client the response is JSON.

```
header('Access-Control-Allow-Methods: GET, POST, PUT, DELETE');
```

- Let browsers know which HTTP methods are allowed in CORS preflight requests.

```
header('Access-Control-Allow-Headers: Content-Type');
```

- Allows the Content-Type header in requests, required for sending JSON.

```
header('Access-Control-Allow-Origin: *');
```

- Enables CORS (Cross-Origin Resource Sharing) from any domain (public API).

What is a preflight Request?

It is an automatic OPTIONS request sent by browsers before certain HTTP requests to check if the actual request is allowed.

```
// Handle preflight OPTIONS request
if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    http_response_code(200);
    exit();
}
```

Browser directly accesses with GET/POST request

- GET or a basic POST are simple requests.
- When browsers make a GET/POSTrequest, they make direct request with simple requests (GET/POST).

Browers requests OPTION otherwise

- If you’re using methods like PUT, DELETE, or custom headers (just like in this example), the browser will first send an OPTIONS request (called a preflight) to ask:

"Hey server, can I send this kind of request?"

- Your server needs to respond with appropriate CORS headers to say “yes.”

What is a “simple” request?

- **Simple** → GET, POST, HEAD + limited headers
  (Content-Type: `text/plain`, `application/x-www-form-urlencoded`, `multipart/form-data`)
- **Non-Simple** → uses `PUT`, `DELETE`, `PATCH`, custom headers, or `application/json`

Browser sends a **preflight** `OPTIONS` request.

Server must reply with headers like:

```
header('Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS');
```

What is Cross-Origin Access (CORS)?

1. For example, the local browser download HTML with JavaScript code from <example.com>.
2. The JavaScript makes the HTTP request (in the browser) to <api.example.com>.
3. PHP (on server) receives the request and sends back a JSON response to <api.example.com>.

Now the browser asks the question if this is an issue or not becuase <example.com> is different from <api.example.com>.

"Does this response say it's OK to share it with code from example.com?"

4. If the response header includes:
   Access-Control-Allow-Origin: http://example.com
   → ✅ JavaScript can access the response
5. If that header is missing:
   → ❌ JavaScript gets blocked from using the data

This is the header to allow CORS.

```
header('Access-Control-Allow-Origin: *');
```

Input Parser

- For the input http://localhost8080/students/123>, the $path becomes "students/123".

```
$method = $_SERVER['REQUEST_METHOD'];
$path = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
$path = trim($path, '/');
```

Split path into segments

- For URL `/students/123`, we need to extract "students" and "123" and store them into variables `$resource = "students"` and `$id = 123`
- We use the explode function to split the path into resource and id.

```php
$segments = explode('/', $path);
$resource = $segments[0] ?? ''; // students
$id = $segments[1] ?? null; // 123
```

null coalescing operator ??

- If the left-hand side is set and not null, use it — otherwise, use the right-hand side.

```
$resource = $segments[0] ?? '';
$id = $segments[1] ?? null;
```

- $resource gets the value of $segments[0], or '' if it's not set or is null.
- $id gets the value of $segments[1], or null if it's not set or is null.

```
$id = $segments[1];          // ❌ This throws a PHP warning if $segments[1] is not set
$id = $segments[1] ?? null; // ✅ This is safe — avoids warning if index is missing
```

Error Processing

- When the URI doesn't have the $resource string, we should respond with the endpoints' information.

```
if (empty($resource)) {
    // Root endpoint - show API info
    echo json_encode([
        'message' => 'Simple Student Management API',
        'endpoints' => [
            'GET /students' => 'Get all students',
            'GET /students/{id}' => 'Get student by ID',
            'POST /students' => 'Create new student',
            'PUT /students/{id}' => 'Update student',
            'DELETE /students/{id}' => 'Delete student'
        ]
    ]);
    exit;
}
```

- We check if the resource name is "students" first; then we get the ID.
- If not, return an error as it doesn't match the API we support.

```
if ($resource === 'students') {
    $student_id = isset($id) ? (int)$id: null;
else {
    http_response_code(404);
    echo json_encode(['error' => 'Resource not found']);
}
```

Routers

- We call the handlers from the $method and $student_id.
- In the error condition, for example, the 'PUT' method without ID, we return an error response.

```
http_response_code(400);
echo json_encode(['error' => 'Student ID required']);
```

```php
switch ($method) {
  case 'GET':
      if ($student_id) { get_student($student_id); }
      else { get_all_students(); }
      break;
  case 'POST':
      create_student();
      break;
  case 'PUT':
      if ($student_id) { update_student($student_id); }
      else {
          http_response_code(400);
          echo json_encode(['error' => 'Student ID required']);
      }
      break;
  case 'DELETE':
      if ($student_id) { delete_student($student_id); }
      else {
          http_response_code(400);
          echo json_encode(['error' => 'Student ID required']);
      }
      break;
  default:
    http_response_code(405);
    echo json_encode(['error' => 'Method not allowed']);
```

Students.php

- We need to make Student class (models/Student.php) to manage students' information effectively.
- The student class has all the information about the student record.
- For each field, we provide a setter and getter.

```
class Student {
    private $id;
    private $name;
    private $email;
    private $major;
    private $year;
    private $created_at;
    private $updated_at;

    public function getId() { return $this->id; }
    public function setId($id) { $this->id = $id; }
```

- The constructor creates a Student object with created/updated information.

```
public function __construct() {
    $this->created_at = date('Y-m-d H:i:s');
    $this->updated_at = date('Y-m-d H:i:s');
}
```

- The toArray function returns a PHP dictionary from the Student object.

```
public function toArray() {
    return [
        'id' => $this->id,
        'name' => $this->name,
        'email' => $this->email,
        'major' => $this->major,
        'year' => $this->year,
        'created_at' => $this->created_at,
        'updated_at' => $this->updated_at
    ];
}
```

handlers.php

- Each handler processes corresponding APIs from the $method.
- We use the sendResponse() function to make a response with a head, code, and a body.

Error Code

- HTTP status codes are grouped into different categories, based on their first digit:
- **1xx – Informational**: Request received and continuing process (e.g., 100 Continue, 101 Switching Protocols).
- **2xx – Success**: The request was successfully received and processed (e.g., 200 OK, 201 Created, 204 No Content).
- **3xx – Redirection**: Further action must be taken to complete the request (e.g., 301 Moved Permanently, 302 Found, 304 Not Modified).

- **4xx – Client Error**: The request contains bad syntax or cannot be fulfilled (e.g., 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 408 Request Timeout, 429 Too Many Requests).
- **5xx – Server Error**: The server failed to fulfill a valid request (e.g., 500 Internal Server Error, 502 Bad Gateway, 503 Service Unavailable, 504 Gateway Timeout)

- Here are some examples from each group:

Code	Meaning
200	OK
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found

Utility Functions

- Load students from JSON file

```
function load_students() {
    $file_path = 'data/students.json';

    if (!file_exists($file_path)) {
        return [];
    }

    $json_data = file_get_contents($file_path);
    $students = json_decode($json_data, true);

    return $students ?: [];
}
```

- Save students to a JSON file

```php
function save_students($students) {
    $file_path = 'data/students.json';
    $json_data = json_encode($students, JSON_PRETTY_PRINT);
    file_put_contents($file_path, $json_data);
}
```

- Get the following available ID

```php
function get_next_id($students) {
    $max_id = 0;
    foreach ($students as $student) {
        if ($student['id'] > $max_id) {
            $max_id = $student['id'];
        }
    }
    return $max_id + 1;
}
```

- This function reads the raw input data and decodes it as JSON.

```php
function getRequestData()
{
    $input = file_get_contents('php://input');
    return json_decode($input, true) ?? [];
}
```

Handlers

- GET /students - Get all students

```
function get_all_students() {
    $students = load_students();
    echo json_encode([
        'success' => true,
        'data' => $students,
        'count' => count($students)
    ]);
}
```

- GET /students/{id} - Get student by ID

```php
function get_student($id) {
    $students = load_students();

    foreach ($students as $student) {
        if ($student['id'] == $id) {
            echo json_encode([
                'success' => true,
                'data' => $student
            ]);
            return;
        }
    }

    http_response_code(404);
    echo json_encode([
        'success' => false,
        'error' => 'Student not found'
    ]);
}
```

- POST /students - Create new student

```php
// Step 1: Get JSON input
function create_student() {
    $input = getRequestData();

    if (!$input) {
        http_response_code(400);
        echo json_encode([
            'success' => false,
            'error' => 'Invalid JSON data'
        ]);
        return;
    }
```

```php
// Step 2: Load students, create a new ID, and create a new student
    // Load existing students
    $students = load_students();

    // Generate new ID
    $new_id = get_next_id($students);

    // Create new student
    $new_student = new Student();
    $new_student->setId($new_id);
    $new_student->setName($input['name'] ?? '');
    $new_student->setEmail($input['email'] ?? '');
    $new_student->setMajor($input['major'] ?? '');
    $new_student->setYear($input['year'] ?? 1);
```

```php
// Step 3:Save and return response
// Add to students array
    $students[] = $new_student->toArray();

    // Save to file
    save_students($students);

    http_response_code(201);
    echo json_encode([
        'success' => true,
        'message' => 'Student created successfully',
        'data' => $new_student->toArray()
    ]);
}
```

- PUT /students/{id} - Update student

```php
function update_student($id)
{
    // Get JSON input
    $input = getRequestData();

    if (!$input) {
        http_response_code(400);
        echo json_encode([
            'success' => false,
            'error' => 'Invalid JSON data'
        ]);
        return;
    }
```

```php
    // Load students
    $students = load_students();

    // Find and update student
    for ($i = 0; $i < count($students); $i++) {
        if ($students[$i]['id'] == $id) {
            // Update fields if provided
            if (isset($input['name'])) $students[$i]['name'] = $input['name'];
            if (isset($input['email'])) $students[$i]['email'] = $input['email'];
            if (isset($input['major'])) $students[$i]['major'] = $input['major'];
            if (isset($input['year'])) $students[$i]['year'] = $input['year'];

            // Update timestamp
            $students[$i]['updated_at'] = date('Y-m-d H:i:s');

            // Save to file & return response
            save_students($students);
            echo json_encode([
                'success' => true,
                'message' => 'Student updated successfully',
                'data' => $students[$i]
            ]);
            return;
        }
    }
```

- If the student is not found, return an error.

```
    http_response_code(404);
    echo json_encode([
        'success' => false,
        'error' => 'Student not found'
    ]);
}
```

- DELETE /students/{id} - Delete student

```php
function delete_student($id)
{
    $students = load_students();

    // Find and remove student
    for ($i = 0; $i < count($students); $i++) {
        if ($students[$i]['id'] == $id) {
            $deleted_student = $students[$i];
            array_splice($students, $i, 1);

            // Save to file
            save_students($students);

            echo json_encode([
                'success' => true,
                'message' => 'Student deleted successfully',
                'data' => $deleted_student
            ]);
            return;
        }
    }

    http_response_code(404);
    echo json_encode([
        'success' => false,
        'error' => 'Student not found'
    ]);
}
```

- The arraysplice() function removes an element in an $student array at $i 1 time.

```
array_splice($students, $i, 1);
//           ↑           ↑  ↑
//           array       start_index  length_to_remove
```

- Why Not Just unset()?

It is known that `unset()` leaves a hole in the array, so what function should we use array_splice function instead when removing array elements.

```
// DON'T do this for deletion:
unset($students[$i]);  // Leaves a "hole" in the array
// DO this instead:
array_splice($students, $i, 1);  // Properly removes and shifts
```

test.html

- Access the 'test.html' from the web browser to check that the Students REST APIs are working correctly.

```
http://localhost:8000/test.html
```

- Make sure the API_BASE matches the server port.

```
    <script>
        const API_BASE = 'http://localhost:8000';
```

HTML/JavaScript

- The HTML div is the placeholder for the button and display.

```
    <div class="test-section">
        <h3>1. Get All Students</h3>
        <button onclick="getAllStudents()">GET /students</button>
        <pre id="all-students-result"></pre>
    </div>
```

- The JavaScript uses the fetch() function to access the REST API server.
- The results are displayed in the HTML document.

```
        async function getAllStudents() {
            try {
                const response = await fetch(`${API_BASE}/students`);
                const data = await response.json();
                document.getElementById('all-students-result').textContent = JSON.stringify(data, null, 2);
            } catch (error) {
                document.getElementById('all-students-result').textContent = 'Error: ' + error.message;
            }
        }
```